\begin{itemize}
    \item Problems are \textbf{due by Wed Feb. 21st before class.}
    \item Problems marked with “P” should be implemented in Haskell and a \textbf{printout} (comment your functions and give sample outputs\(^1\)) should be turned as part of the assignment.
    \item If you turn in more than one page, \textbf{staple} all of those pages together!!!
\end{itemize}

\textbf{(Small) Group Assignment 2}

\textbf{Problem 1 (P, ADT Queue)}

Define an abstract data type \textit{Queue} with the following operations:

\begin{itemize}
    \item \textit{newQ}: returns a new empty queue
    \item \textit{emptyQ}: given a queue \( q \), returns \textbf{True} if \( q \) is empty and \textbf{False} otherwise
    \item \textit{enQ}: adds an element to the \textit{end} of the queue and returns the new queue
    \item \textit{deQ}: removes an element from the \textit{front} of the queue and returns the new queue
    \item \textit{headQ}: returns the first element of the queue
\end{itemize}

\begin{enumerate}
    \item \textit{Specification}: (i) give the \textit{signatures} of these operations in Haskell (make \textit{Queue} a polymorphic type), and (ii) give the \textit{axioms} (equations) that your \textit{Queue} ADT should observe.
    
    \item \textit{Implementation}: in Haskell, (i) define the concrete (and polymorphic) data type for \textit{Queue}, and (ii) define the abovementioned functions.
    
    \item As a test output, print the value for \texttt{foldr enQ newQ [1..10]}. (Do you see what this does??)
\end{enumerate}

\textbf{Problem 2 (P, List Reversal using Stack)}

\begin{enumerate}
    \item Define the polymorphic ADT \textit{Stack} in Haskell: the specification can be as given in class, but the implementation should be based on Haskell lists.
    
    \item Use your \textit{Stack} ADT to implement a function \texttt{reverse :: [a] \rightarrow [a]} that reverses a list by pushing all elements of the given list on a stack and creates the reversed list by popping the elements from the stack.
\end{enumerate}

\textbf{Problem 3 (Structural Induction)}

The \textit{height} of a tree is defined as the length of the longest path from the root to any leaf.

\begin{enumerate}
    \item Define a Haskell function \texttt{myheight} that works on a data type \texttt{BinTree} for binary trees and that returns the height of the tree.
    
    \item Prove the correctness of your implementation \texttt{myheight}, i.e., use structural induction to show that for any \( bt \in \texttt{BinTree} \) we have: \( \text{height}(bt) = \text{myheight}(bt) \).
\end{enumerate}

\(^1\)don’t try to fake those outputs if you don’t have the solution; rather explain what you have and where the problem is!